tank_01

预备：没有下载JavaDoc的到Oracle网站下载
新建项目，将rt.jar关联到下载的JavaDoc

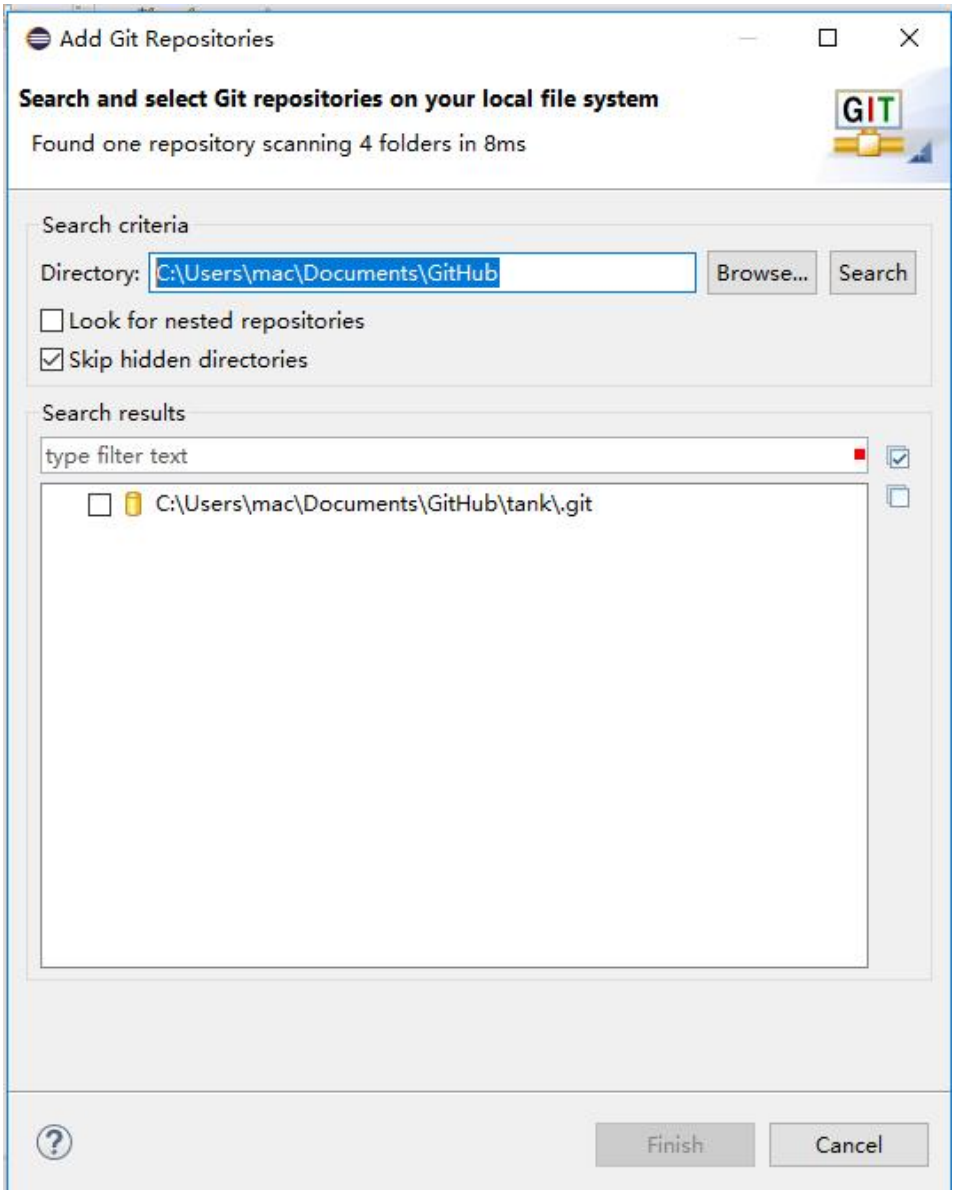
学编程的不二法门：敲
代码量代表一切

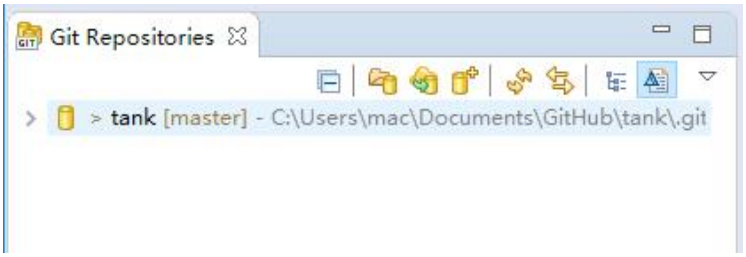
1. 注册github
2. desktop.github.com 下载github windows 并安装
3. clone repository
4. 填写项目bjmashibing/tank
5. local path / 空目录

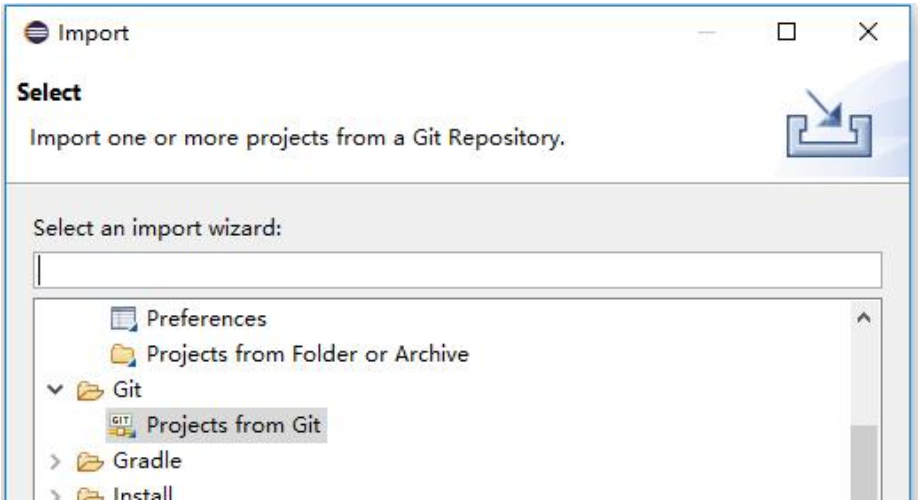
第一天作业：
根据四个boolean值，计算坦克方向，根据坦克方向和速度，自动移动位置。（假设坦克不能停）

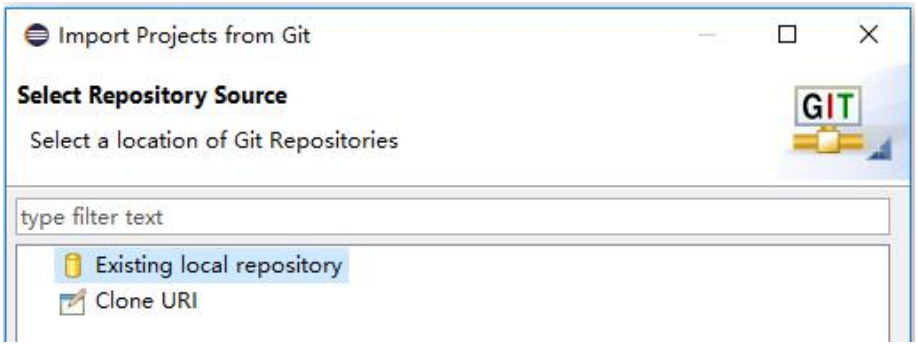
eclipse引入git项目:

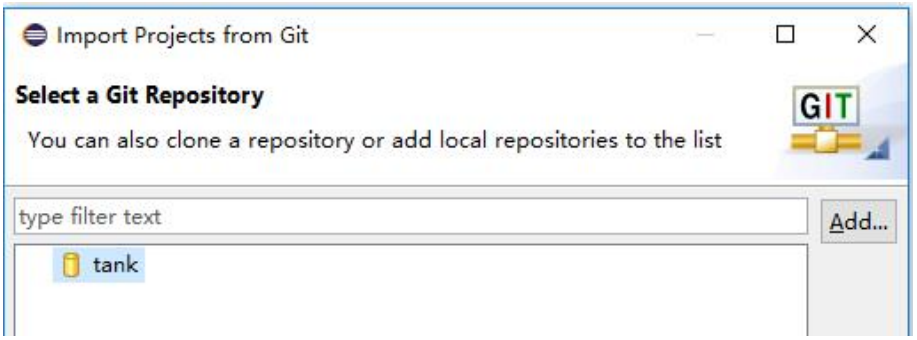
window - show view - git repositories
add an existing local git repo


Add Git Repositories
Search and select Git repositories on your local file system
Found one repository scanning 4 folders in 8ms
GIT
Search criteria
Directory: C:\Users\mac\Documents\GitHub
Browse...
Search
Look for nested repositories
Skip hidden directories
Search results
type filter text
C:\Users\mac\Documents\GitHub\tank\.git
Finish
Cancel


Git Repositories
tank [master] - C:\Users\mac\Documents\GitHub\tank\.git







然后一路next

引入如果有问题的，就删掉本地库，然后重新从github克隆

tank_02

1. 如何定义主战坦克的方向
    a. Enum Dir
2. 根据按键改变主战坦克方向
    a. setMainTankDir()
3. 根据方向进行坦克的移动
4. 怎么 样处理坦克静止状态

a. moving = false;

5. 想象如何给出更多坦克，以及子弹
    a. 将坦克封装成类，理解面向对象设计中“封装”的思想
6. 用双缓冲解决闪烁问题（不重要）
    a. repaint - update
    b. 截获update
    c. 首先把该画出来的东西（坦克， 子弹）先画在内存的图片中，图片大小和游戏画面一致
    d. 把内存中图片一次性画到屏幕上（内存的内容复制到显存）
7. 打出一颗子弹
    a. 按下Ctrl键，主战坦克打出一颗子弹
    b. 用面向对象的思想考虑
8. 打出一串子弹
    a. 将子弹装在容器中

作业：搞出一个排的敌人的坦克，挨排儿干掉他们

tank_03

1. 将坦克换成图片
    a. 关于classloader的基础知识
    b. 显示图片，使用ImageIO
2. 将子弹换成图片
3. 将子弹从坦克中心位置打出
    a. 根据坦克图片的大小，和左上角的位置计算子弹左上角的位置
4. 加入敌军坦克
    a. 分拨儿Group
5. 子弹与敌军坦克的碰撞检测
    a. 击毁一辆坦克
6. 加入多辆敌军坦克
7. 让敌军坦克动起来
8. 加入爆炸
9. 加入声音

作业：

1. 把爆炸功能完成

2. 贪吃蛇 打飞机
3. 敌人坦克随机动，随机发射子弹
4. 边界检测

tank_04

1. 调整效果
    a. 换坦克图片，换炮弹图片
    b. 音乐与音效
2. 完成爆炸功能
    a. 加入到List中
    b. 显示数量
3. 敌人坦克简单智能化
    a. 随机移动位置
    b. 随机发射子弹
4. 边界检测
5. 解决new Rectangle问题
6. MileStone - 里程碑式的版本
7. Code Review and Refactoring
8. 配置文件
9. PropertyMgr的单例问题
10. 答疑
    a. java - exe jar+jre+tool = exe

作业：
整理代码
写出首个设计模式（单例）
在内存中，如果需要某个类的对象，在程序上保证，有且只有一个该类的对象
策略模式

tank_05

坦克从今日始，分支两个：DesignPatterns Network

1. 单例：PropertyMgr ResourceMgr
2. 策略：Strategy
    a. Comparable
    b. Comparator

作业：
策略模式应用到：Tank.fire

策略1：默认策略 一颗子弹

策略2：四个方向同时打出子弹

策略3：打核弹

.....

fire(FireStrategy s) -> 每次调用，都需要new，因此，应该把DefaultStrategy -> Singleton 成员变量

Extensibility Scalability

tank_06

1. 工厂系列
2. 简单工厂
3. 静态工厂
4. 工厂方法FactoryMethod
    a. 产品维度扩展
5. 抽象工厂
    a. 产品一族进行扩展

作业：用抽象工厂，完成一键风格替换

maven :

project - add framework support - maven

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-context</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
</dependencies>
```

import changes / auto import

tank_07

1. SpringIOC
2. Facade
    a. 门面
3. Mediator
    a. 调停者
4. Decorator
    a. 装饰器
5. 责任链开头

TF - 解决添加新游戏物体的问题

1：TF - Facade

Frame - > 展示

GameModel -> 内部逻辑计算

2：GameObject

spring配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-
beans.xsd">
```

```
    <bean id="d" class="com.mashibing.Driver"></bean>
    <bean id="tank" class="com.mashibing.Tank">
        <property name="driver" ref="d"></property>
    </bean>

</beans>
```

spring的写法：

```
public class Main {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("app.xml");

        //Driver d = (Driver)context.getBean("driver");
        Tank t = (Tank)context.getBean("tank");

    }
}
```

tank_08

1. Facade
2. Mediator
3. 完成坦克责任链模式
4. Observer

作业：

1：colliders 写到配置文件

2：加新的游戏物体，墙，不需要修改原来任何一行代码

tank_09

1. 代码重构
    a. tank.back()

i. 用oldX, oldY记录上一步的位置，坦克互相撞击后回到上一步

ii. 修改TankTankCollider

b. 添加一堵墙

i. class Wall extends GameObject

ii. 定义属性rect

iii. 定义BulletWallCollider

iv. 定义TankWallCollider

c. bullet.CollideWithTank()的逻辑移到BulletTankCollider中

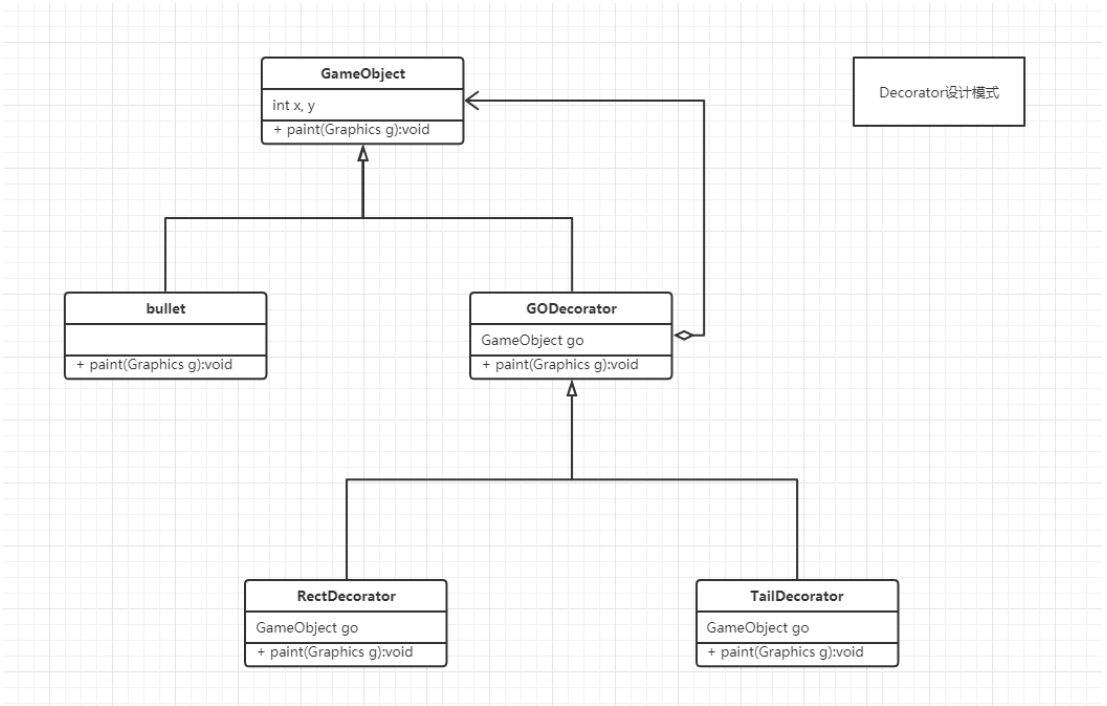
d. GameModel做成单例

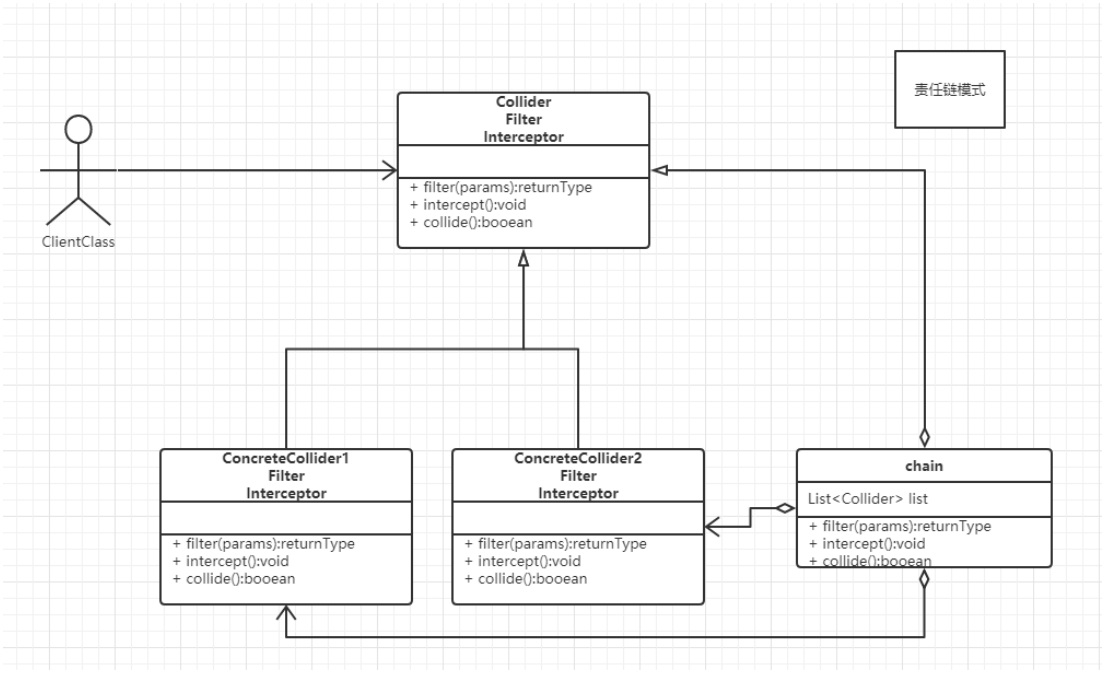
e. 对于GameModel.add()方法的优化

i. 考虑消除new GameModel需要new Tank, new Tank又需要new GameModel

ii. GameModel . init();

2. 详解Decorator设计模式





3.

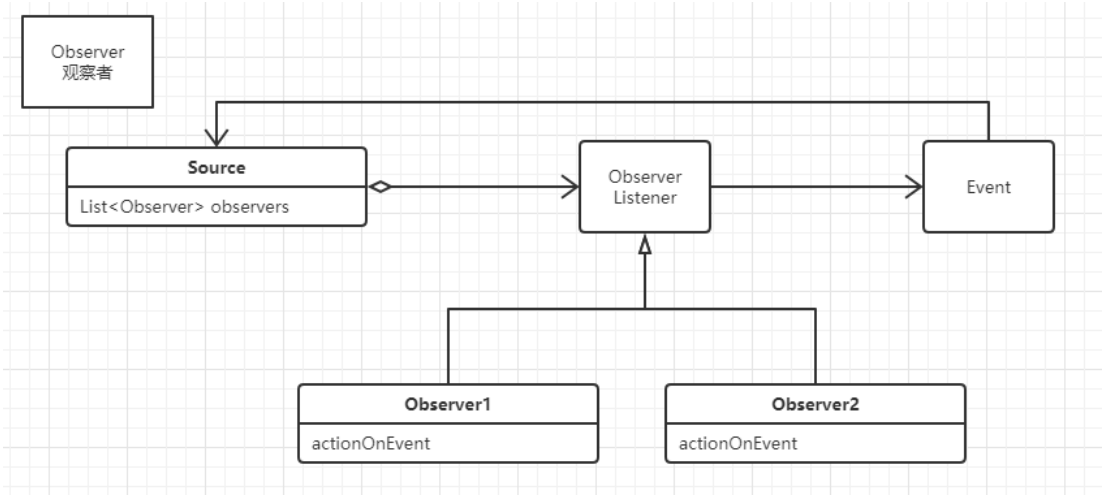
4. Flyweight?

作业：

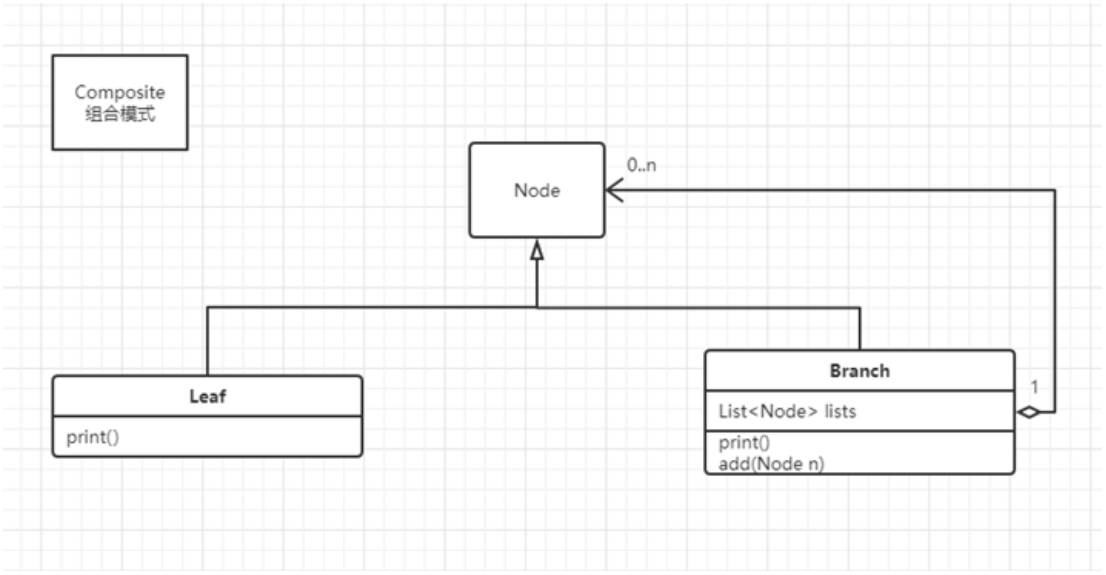
敲代码

tank_10

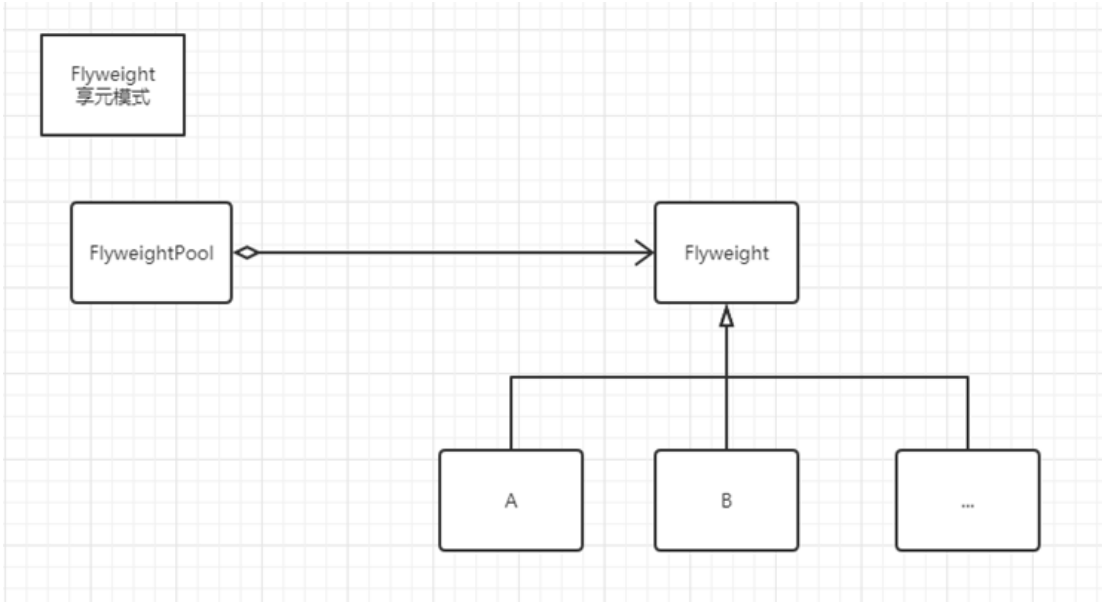
1. 详解Observer设计模式



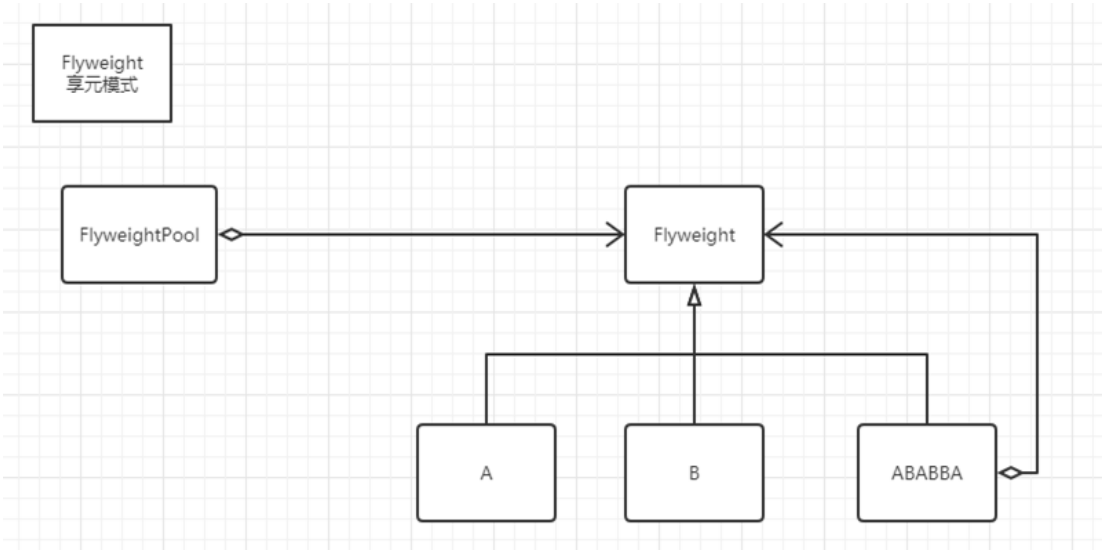
2. Composite



3. Flyweight



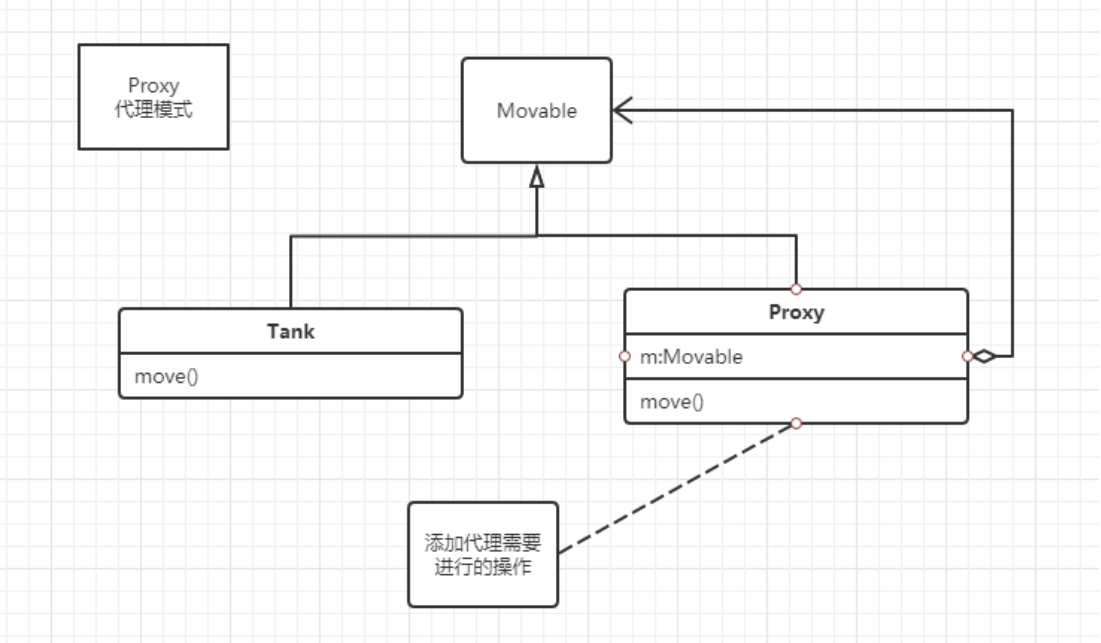
结合composite



tank_11

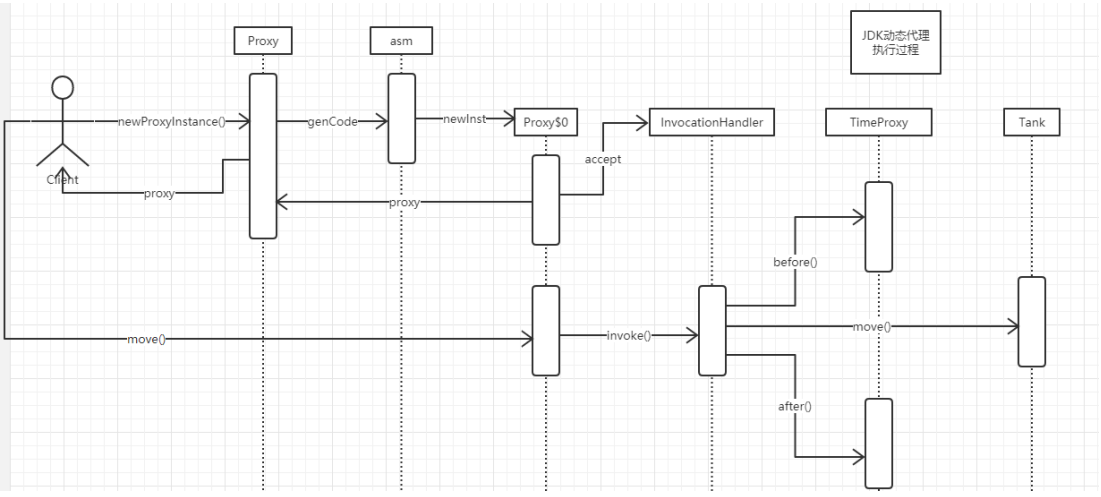
1. 代理模式

    a. 静态代理



    b. 动态代理

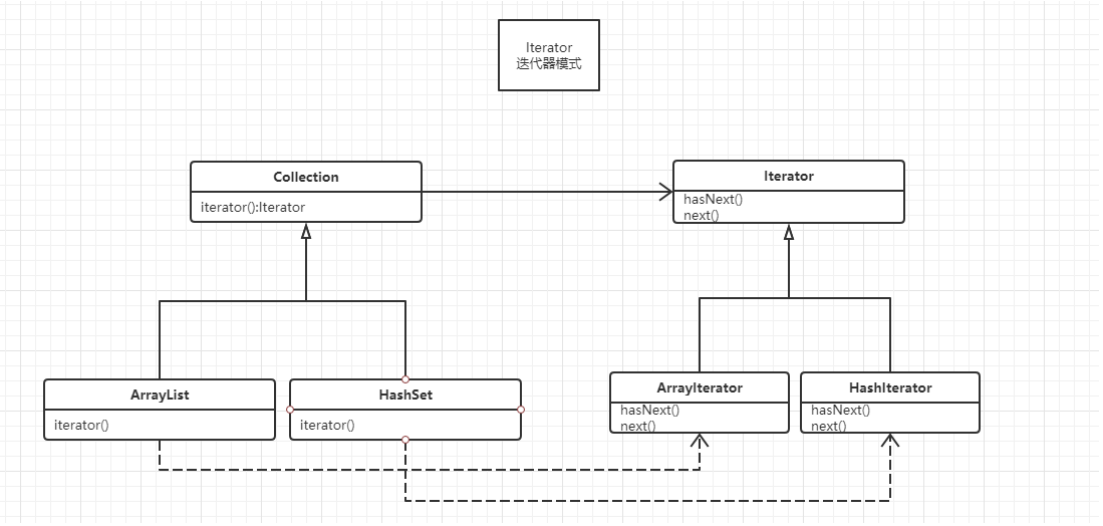
1. JDK实现动态代理



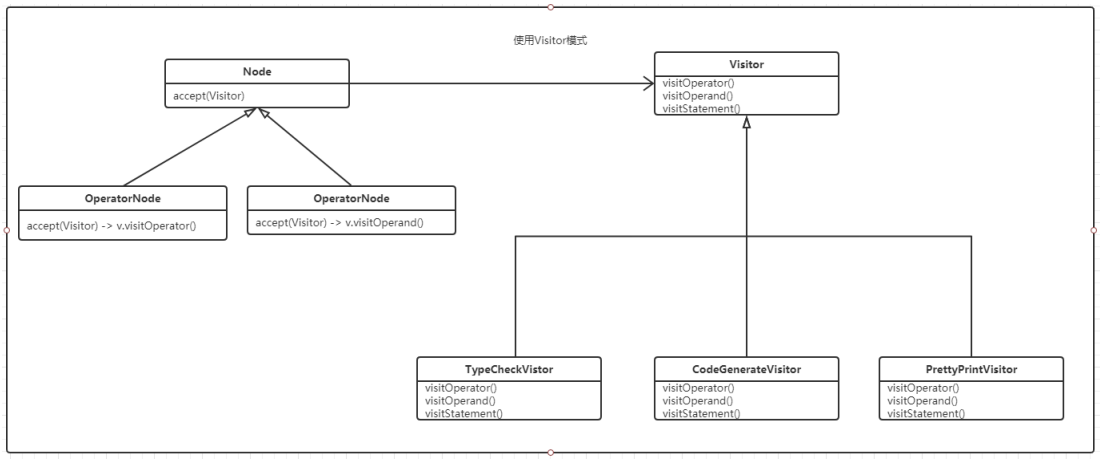
3. CGLIB实现动态代理
4. SpringAOP

tank_12

1. Iterator



2. Visitor



3. ASM？(JVM细节 class类文件的格式）繁琐 + 枯燥

a. iterator + vistor + chainofresposibility

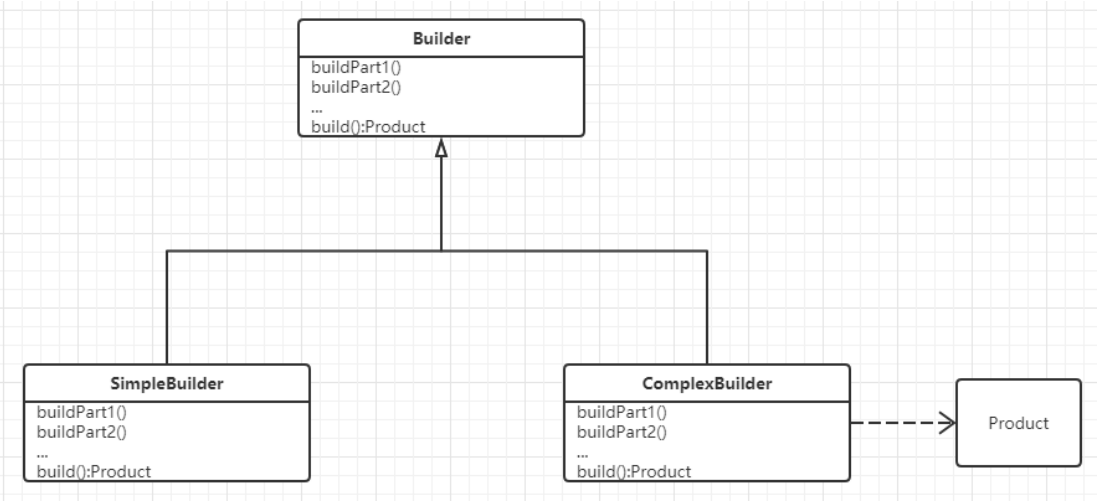
作业：实现LinkedList的Iterator

tank_13

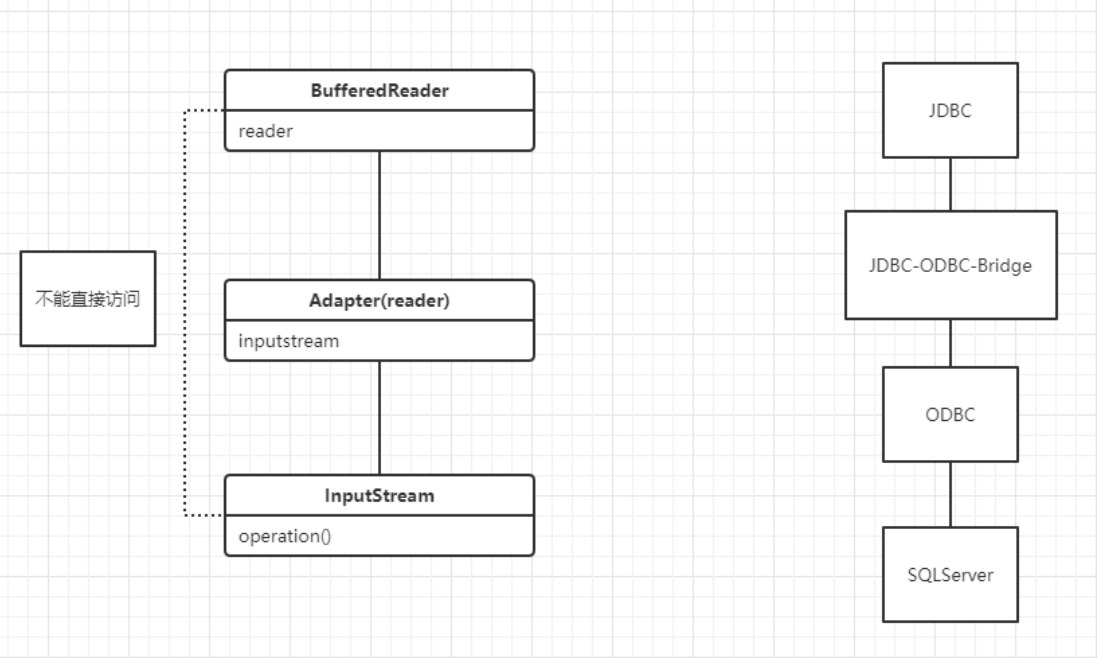
1. COR的作业

a. 模拟Servlet的FilterChain

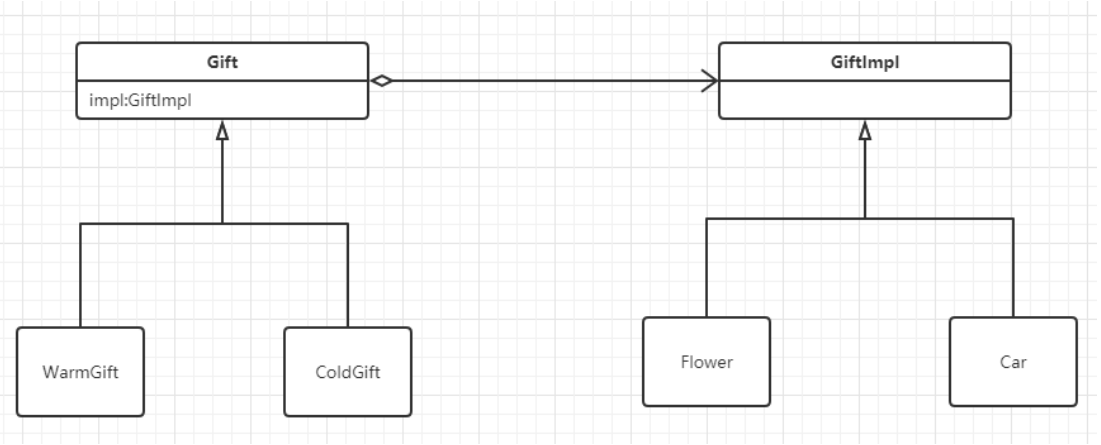
2. Builder



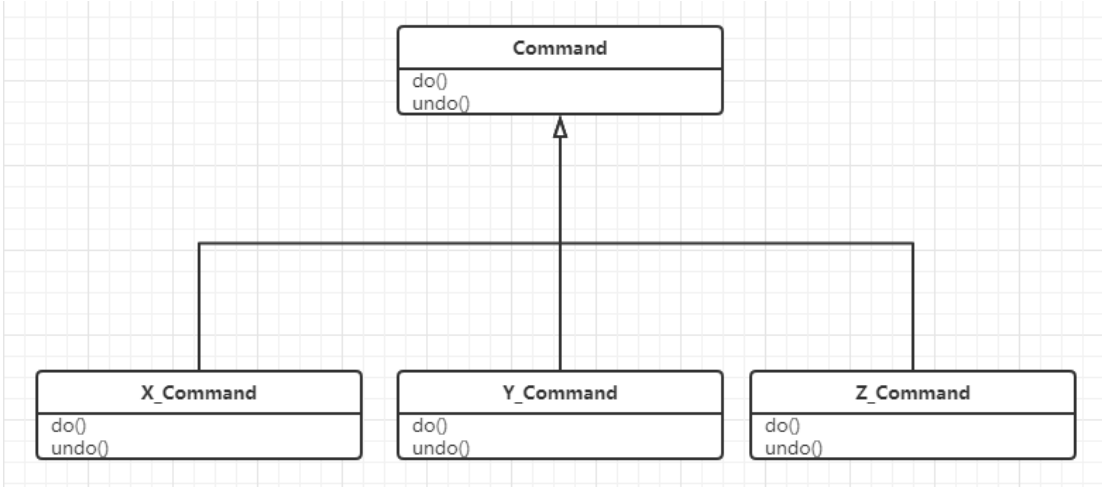
3. Adapter



4. Bridge



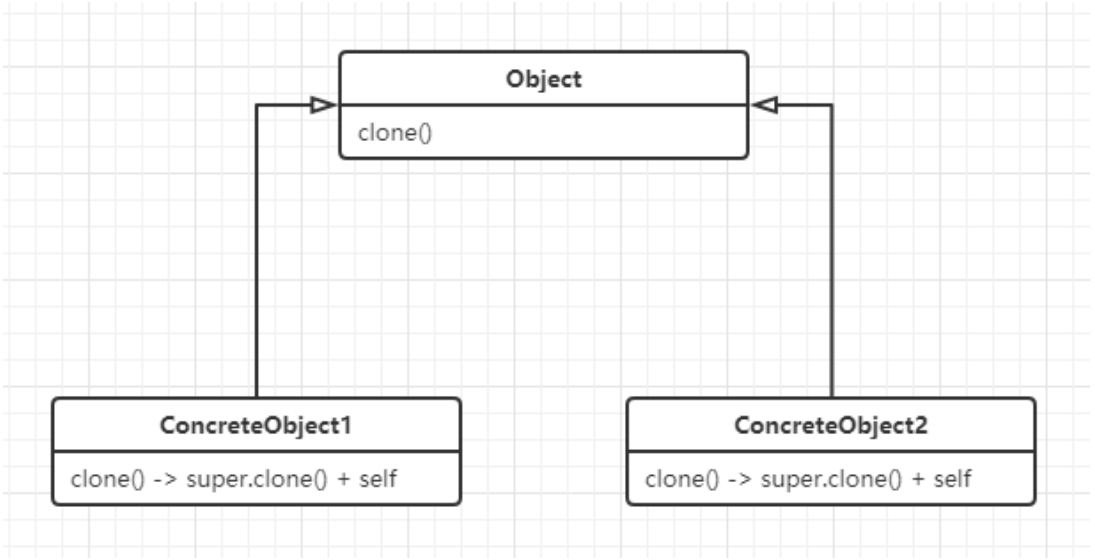
5. Command



6. 作业：command + cor

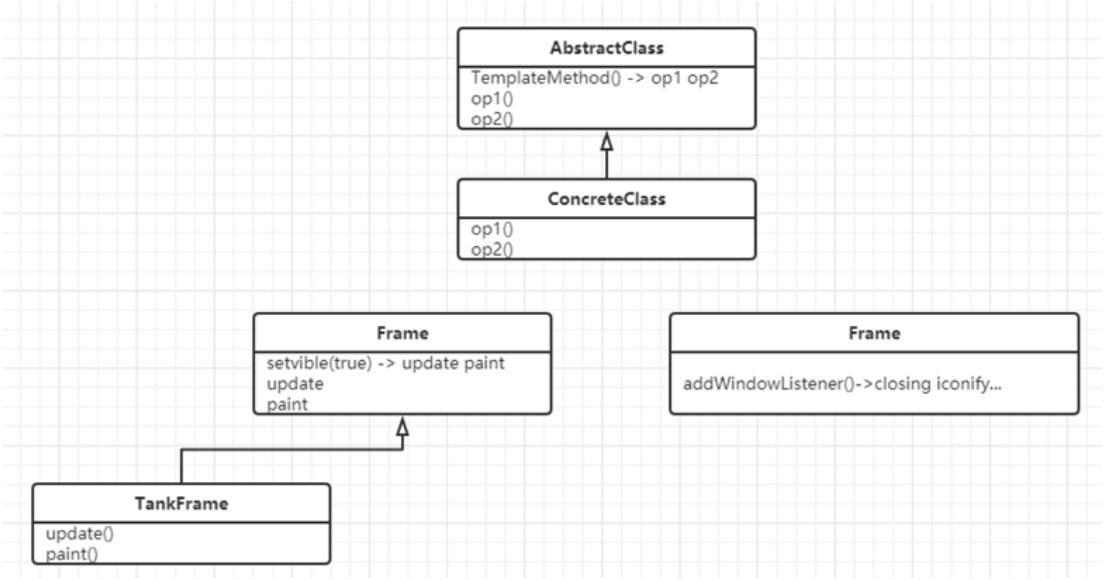
tank_14

1. 作业：
    a. command + cor
2. prototype -> Object.clone()

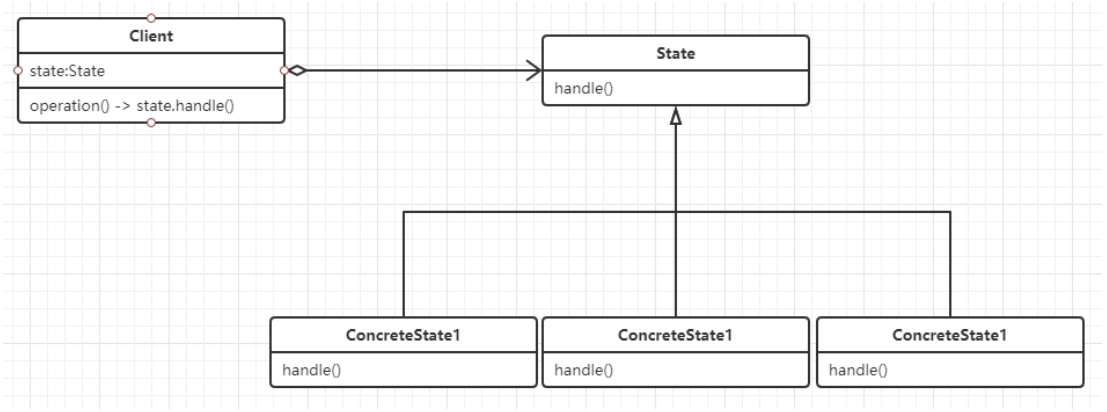


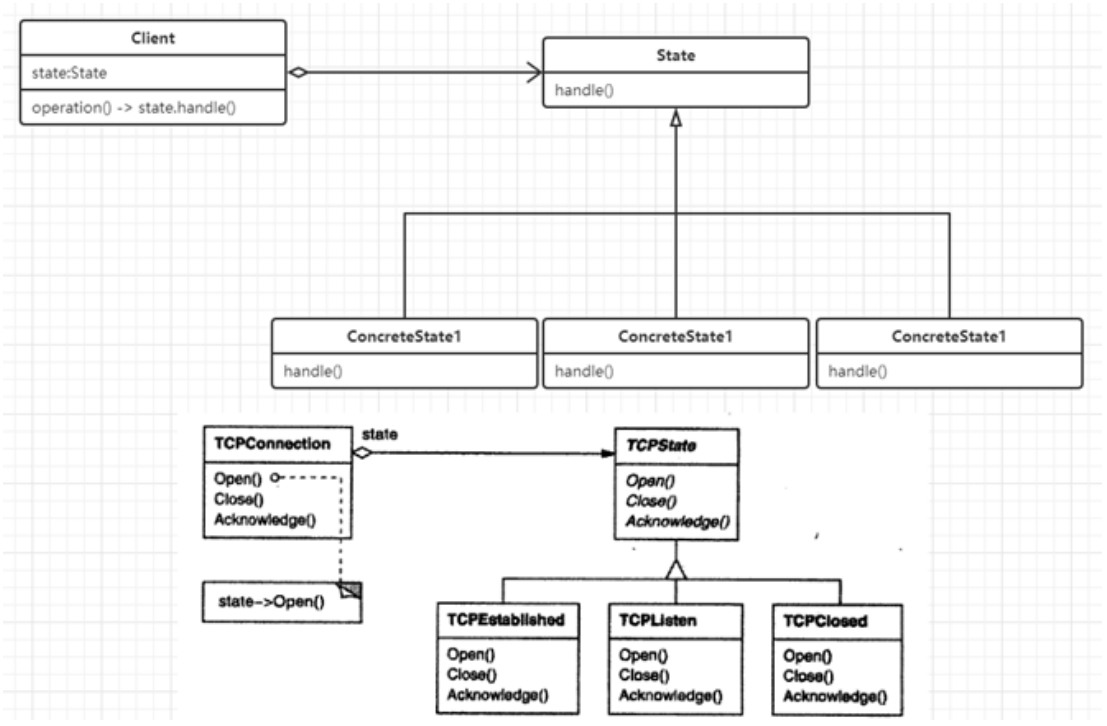
3. memento -> java.io.Serializable

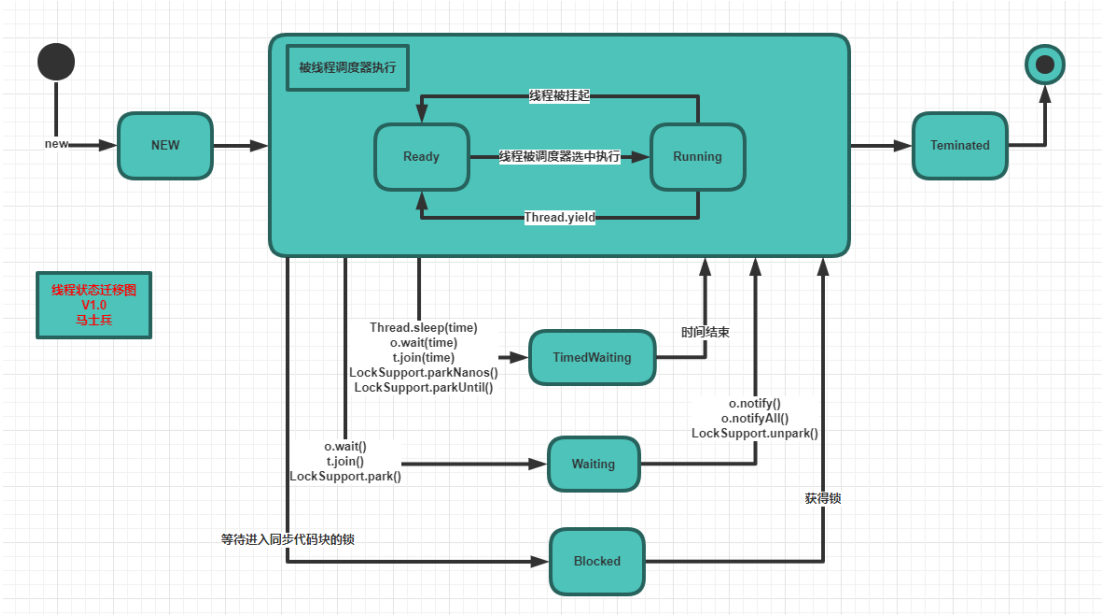
4. TemplateMethod


AbstractClass
TemplateMethod() -> op1 op2
op1()
op2()
ConcreteClass
op1()
op2()
Frame
setvible(true) -> update paint
update
paint
Frame
addWindowListener()->closing iconify...
TankFrame
update()
paint()

5. State


Client
state:State
operation() -> state.handle()
State
handle()
ConcreteState1
handle()
ConcreteState1
handle()
ConcreteState1
handle()


Client
state:State
operation() -> state.handle()
State
handle()
ConcreteState1
handle()
ConcreteState1
handle()
ConcreteState1
handle()
TCPConnection
Open()
Close()
Acknowledge()
state
TCPState
Open()
Close()
Acknowledge()
state->Open()
TCPEstablished
Open()
Close()
Acknowledge()
TCPListen
Open()
Close()
Acknowledge()
TCPClosed
Open()
Close()
Acknowledge()



6. Intepreter
    a. （暂略）

作业：

State/Action	open the door	close the door	run the car	stop the car
Open	✘	✓	✘	✘
Closed	✓			
Running				
Stopped				

tank_15_16

1. 为网络版坦克做好准备，学习真正生产环境中的TCPServer的写法
    a. 学习使用JDK Loggin API
        i. https://www.cnblogs.com/liaojie970/p/5582147.html
    b. 复习BIO-NIO-AIO-Netty的编程模型
        i. 网络程序的烦点 异常处理 正确关闭 ->线程的正常结束 ->线程池的正常结束
    c. 写一个NettyClient
        i. EventLoopGroup 网络IO事件处理线程组
        ii. Netty中的任何方法都是异步模型
        iii. 首先使用ChannelInitializer.initChannle() 添加channel的handler

iv. 在channelHandler中处理业务逻辑，学习使用ChannelInboundHandlerAdapter

d. 写一个NettyServer

i. nettycient传递数据给nettyserver

ii. ByteBuf的使用

iii. 保存多个客户端

iv. 接收到一个客户端的数据后传递给多个客户端

e. 写一个图形化的Client，顺带完成聊天

f. 写一个图形化的NettyServer

g. 将NettyServer能够优雅的关闭

h. 将项目转化为Maven项目

i. 项目右击 - Configure - Convert to Maven Project

ii. 在Maven中添加依赖：io.netty:netty-all:4.1.9.Final

iii. 如果出现java.lang.Object无法解析问题，重置buildpath

2. check out master
3. 去掉NPC Non-Player Character
    a. Main.java 敌人不再初始化
4. 为了在网络端区分坦克，在Tank中加入UUID属性
5. 创建net package
6. 创建net.Server类
    a. 为了调试程序，将Server做成窗口类

```
public class Server extends Frame {

 Button btnStart = new Button("start");
 TextArea taLeft = new TextArea();
 TextArea taRight = new TextArea();

 public Server() {
 this.setSize(1600, 600);
 this.setLocation(300, 30);
  this.add(btnStart, BorderLayout.NORTH);
  Panel p = new Panel(new GridLayout(1, 2));
  p.add(taLeft);
  p.add(taRight);
  this.add(p);
  this.addWindowListener(new WindowAdapter() {
  @Override
  public void windowClosing(WindowEvent e) {
```

```
18   System.exit(0);
19   }
20   });
21   this.setVisible(true);
22   }
23
24   public static void main(String[] args) {
25   new Server();
26   }
27  }
```

7. 添加辅助方法：addLeft addRight

```
1  public void addLeft(String txt) {
2   taLeft.setText(taLeft.getText() + txt +
3   System.getProperty("line.separator"));
4  }
```

8. 加入对于btnStart的Action处理

```
1  btnStart.addActionListener(new ActionListener() {
2   @Override
3   public void actionPerformed(ActionEvent e) {
4   button.setEnabled(false);
5   button.setLabel("started");
6   try {
7   Server.this.start();
8   } catch (Exception e1) {
9   e1.printStackTrace();
10   }
11   }
12   });
```

9. 完成start方法
    a. 创建bossGroup和workerGroup
    b. 创建ServerBootstrap
    c. 加入ChannelInitializer的处理
    d. 每一个Channle addLast(new ServerHandler)
10. 完成ServerHandler，重写方法
    a. channelRead()
    b. exceptionCaught()
11. Client端编程
    a. 连接并发送一个字符串
    b. 服务器端接收并返回
    c. 客户端接收
12. 正确的结束程序

13. 自定义消息 使用Netty Codec API

tank_17

1. checkout master
    a. 项目右键 - team - switch to - master
2. 改造现有程序
    a. 去掉敌人坦克 注释掉main里面初始化敌方坦克的内容
    b. 复制Client Server ServerFrame TankMsg TankMsgDecoder TankMsgEncoder
    c. 引入Netty依赖
        i. 将项目转换为Maven项目
        ii. 添加依赖 io.netty:netty-all:4.1.9.Final
        iii. 这时候有可能出现jre lib依赖丢失的问题，在项目上右击 - build path - add lib - 选择jre重新加入
    d. 注释掉客户端ClientFrame的部分
3. 将TankMsg修改为TankJoinMsg
    a. 将tank加入UUID属性 标识独一无二的id号
    b. 写单元测试！！！！重要！！！！
4. 写通TankJoinMsg环路
    a. 阅读源码的历史版本
5. 接收到TankJoinMsg的逻辑处理
    a. 是不是自己?
    b. 列表是不是已经有了
    c. 发自己的一个TankJoinMsg
6. 代码重构
    a. 用HashMap代替LinkedList
    b. 把Client设定为单例
    c. 在TankJoinMsg中处理接收
7. 处理多种消息（作业）
    a. 分析会有哪些消息?
    b. 处理各种各样的消息。

tank_18

1. 加入Msg体系后的代码修正
    a. 加入重写方法
    b. 加入MsgType类
    c. Client.send(改为Msg类型)
    d. ChannelRead0 改为Msg类型
    e. Encoder和Decoder
    f. 修改单元测试
2. 加入新的TankStartMovingMsg
    a. 修改TankJoinMsgDecoder为MsgDecoder Encoder同样处理
    b. 加入TankStartMovingMsg 并重写相应方法
    c. 加入TankStartMovingMsg(Tank)构造方法
    d. 修改ClientHandler channelRead0 处理Msg
    e. 在setMainDir中发出TankStartmoving消息
    f. 在TankStartMoving.handle()中处理收到该消息的逻辑
3. 加入TankStopMsg
    a. 何时发出
    b. 解码处理
    c. 编解码测试
    d. 收到如何处理
4. 作业：
    a. 小Bug，发了太多的TankStartMoving的消息
    b. 加入其它必要的Msg，完成基本的联网功能
        i. 子弹的消息
        ii. 坦克死掉的消息

设计模式列表

创建型模式...

1. Abstract Factory..
2. Builder

3. Factory Method..
4. Prototype..
5. Singleton.

结构型模式...

1. Adapter.
2. Bridge..
3. Composite..
4. Decorator.
5. Facade..
6. Flyweight.
7. Proxy..

行为模式...

1. Chain of Responsibility.
2. Command.
3. Interpreter.
4. Iterator
5. Mediator
6. Memento..
7. Observer
8. State..
9. Strategy.
10. Template Method..
11. Visitor.